

**IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
WACO DIVISION**

LOWER48 IP LLC,

Plaintiff,

v.

SHOPIFY, INC.,

Defendant.

Case No. 6:22-cv-00997-DAE

Jury Trial Demanded

**FIRST AMENDED COMPLAINT FOR
PATENT INFRINGEMENT**

Plaintiff Lower48 IP LLC (“Lower48”) files this First Amended Complaint against Shopify, Inc. (“Shopify”) for patent infringement of United States Patent Nos. 7,681,177, 10,140,349, 11,100,070, and 11,194,777 (collectively the “patents-in-suit”) and alleges as follows:

NATURE OF THE ACTION

1. This is an action for patent infringement arising under the patent laws of the United States, 35 U.S.C. §§ 1 *et seq.*

THE PARTIES

2. Plaintiff Lower48 is a limited liability company organized under laws of the State of Texas with its principal place of business at 8226 Douglas Avenue, Suite #325, Dallas, Texas 75225.

3. Lower48 is affiliated with Lower48 Analytics, Inc., a data analysis and

portfolio management services provider to the oil and gas industry.¹

¹ See e.g., <https://www.lower48.com>.

4. On information and belief, Defendant Shopify is a Canadian company with a principal address at 150 Elgin Street, 8th Floor, Ottawa, ON K2P 1L4, Canada.

JURISDICTION AND VENUE

5. This Court has subject matter jurisdiction over this action pursuant to 28 U.S.C. §§ 1331 and 1338(a) because this action arises under the patent laws of the United States, 35 U.S.C. §§ 1 *et seq.*

6. On information and belief, Shopify is subject to this Court's personal jurisdiction, in accordance with due process and/or the Texas Long Arm Statute because, in part, Shopify has committed, and continues to commit, acts of infringement in the State of Texas, has conducted business in the State of Texas, and/or has engaged in continuous and systemic activities in the State of Texas.

7. On information and belief, Shopify offers a variety of products and services that rely, utilize, and/or employ in whole or in part functionality relating to the GraphQL query language, including without limitation, Shopify's GraphQL app, Logo Maker, Business Name Generator, Slogan Maker, Domain Name Generator, Link in Bio for Commerce, WHOIS Domain Lookup, Wholesale Marketplace, Pay Stub Generator, QR Code Generator, Terms and Conditions Generator, Business Card Maker, Invoice Generator, Purchase Order Template, Privacy Policy Generator, Shipping Label Template, Refund Policy Generator, Bill of Lading Template, Barcode Generator, Image Resizer, Business Loan Calculator, CPM Calculator, Profit Margin Calculator, Exchange Marketplace, Discount Calculator, products and services

incorporating in whole or in part Shopify's Admin API,² products and services incorporating in whole or in part Shopify's Storefront API,³ products and services incorporating in whole or in part Shopify's Partner API,⁴ products and services incorporating in whole or in part Shopify's Payment Apps API,⁵ products and services incorporating in whole or in part Shopify's Marketplaces API,⁶ products and services incorporating in whole or in part Shopify's ShopifyQL API,⁷ among other Shopify products and service offerings which operate in a substantially similar manner to the above referenced examples. (Hereinafter, referred to collectively and individually as the "Shopify GraphQL Products and Services".)

8. On information and belief, the accused Shopify GraphQL Products and Services infringe one or more claims of each of the patents-in suit.

9. On information and belief, Shopify provides the Shopify GraphQL Products and Services, as a result of its unauthorized making, use, sale, importation, offers to sell, sale of access, and/or offers to sell access to mobile application software, website access, web servers, API servers, database servers, servers, data storage devices, front-end software, back-end hardware, and back-end software. (Hereinafter referred to collectively and individually as the "Shopify GraphQL System".)

10. On information and belief, the accused Shopify GraphQL System

² See e.g., <https://shopify.dev/api/admin>.

³ See e.g., <https://shopify.dev/api/storefront>.

⁴ See e.g., <https://shopify.dev/api/partner>.

⁵ See e.g., <https://shopify.dev/api/payments-apps>.

⁶ See e.g., <https://shopify.dev/api/marketplaces>.

⁷ See e.g., <https://shopify.dev/api/shopifyql>.

infringes one or more claims of each of the patents-in suit.

11. This Court has personal jurisdiction over Shopify because it committed and continues to commit acts of infringement in this judicial district in violation of 35 U.S.C. §§ 271(a). In particular, on information and belief, Shopify has made, used, imported, offered to sell and/or sold infringing products, services and/or systems in this judicial district, including the accused Shopify GraphQL Products and Services and Shopify GraphQL System.

12. On information and belief, Shopify is subject to the Court's jurisdiction because it regularly conducts and solicits business, or otherwise engages in other persistent courses of conduct in this district, and/or derives substantial revenue from the sale and distribution of goods and services provided to individuals and businesses in this district.

13. This Court has personal jurisdiction over Shopify because, *inter alia*, Shopify, on information and belief: (1) has committed acts of patent infringement in this judicial district, (2) has substantial, continuous, and systematic contacts with this State and this judicial district; (3) owns, manages, and operates facilities in this State and this judicial district; (4) enjoys substantial income from its operations and sales in this State and this judicial district; (5) employs Texas residents in this State, and (6) solicits business and markets products, systems and/or services in this State and judicial district including, without limitation, the infringing Shopify GraphQL Products and Services and Shopify GraphQL System.

14. Venue is proper pursuant to 28 U.S.C. §§ 1391(c)(3) because Shopify is a

foreign corporation not residing in a United States judicial district, and, therefore, may be sued in any judicial district pursuant to 28 U.S.C. §§ 1391(c)(3).

United States Patent No. 7,681,177

15. On March 16, 2010, the United States Patent and Trademark Office (“USPTO”) duly and legally issued United States Patent No. 7,681,177 (“the ‘177 patent”) entitled “Method and/or System for Transforming Between Trees and Strings” to inventor Jack J. LeTourneau.

16. The ‘177 patent is presumed valid under 35 U.S.C. § 282.

17. Lower48 owns all rights, title, and interest in the ‘177 patent.

18. Lower48 has not granted Shopify a license to the rights under the ‘177 patent.

19. The ‘177 patent relates to, among other things, improvements to the functionality of database systems.

20. The claimed invention(s) of the ‘177 patent sought to solve problems with, and improve upon, existing database systems. For example, the ‘177 patent states:

In a variety of fields, data or a set of data, may be represented in a hierarchical fashion. This form of representation may, for example, convey information, such as particular relationships between particular pieces of data and the like. However, manipulating such data representations is not straight-forward, particularly where the data is arranged in a complex hierarchy. Without loss of generality, one example may include a relational database. Techniques for performing operations on such a database, for example, are computationally complex or otherwise cumbersome. A continuing need, therefore, exists for additional techniques for manipulating data hierarchies.

See ‘177 Specification at col. 1, ll. 18-29.

21. The '177 patent then states:

In a variety of fields, data or a set of data, may be represented in a hierarchical fashion. This form of representation may, for example, convey information, such as particular relationships or patterns between particular pieces of data or groups of data and the like. However, manipulating and/or even recognizing specific data representations or patterns is not straight-forward, particularly where the data is arranged in a complex hierarchy. Without loss of generality, examples may include a database and further, without limitation, a relational database. Techniques for performing operations on such databases or recognizing specific patterns, for example, are computationally complex, time consuming, and/or otherwise cumbersome. A need, therefore, continues to exist for improved techniques for performing such operations and/or recognizing such patterns.

See '177 Specification at col. 2, ll. 47-61.

22. The invention(s) claimed in the '177 patent solves various technological problems inherent in the then-existing database systems and enables database systems to, among other things, (1) function more efficiently, (2) improve management and/or utilization of disparate data sources, (3) reduce data retrieval latency, (4) improve data processing speeds and/or flexibility, (5) reduce data processing and/or transmission overhead, and (6) improve ease of use.

United States Patent No. 10,140,349

23. On November 27, 2018, the USPTO duly and legally issued United States Patent No. 10,140,349 ("the '349 patent") entitled "Method and/or System for Transforming Between Trees and Strings" to inventor Jack J. LeTourneau.

24. The '349 patent is presumed valid under 35 U.S.C. § 282.

25. Lower48 owns all rights, title, and interest in the '349 patent.

26. Lower48 has not granted Shopify a license to the rights under the '349 patent.

27. The '349 patent relates to, among other things, improvements to the functionality of database systems.

28. The invention(s) claimed in the '349 patent is a continuation of the '177 patent and solves the same technological challenges as the '177 patent.

United States Patent No. 11,100,070

29. On August 24, 2021, the USPTO duly and legally issued United States Patent No. 11,100,070 ("the '070 patent") entitled "Manipulation and/or Analysis of Hierarchical Data" to inventors Karl Schiffmann, Mark Andrews, and Jack J. LeTourneau.

30. The '070 patent is presumed valid under 35 U.S.C. § 282.

31. Lower48 owns all rights, title, and interest in the '070 patent.

32. Lower48 has not granted Shopify a license to the rights under the '070 patent.

33. The '070 patent relates to, among other things, improvements to the functionality of database systems.

34. The claimed invention(s) of the '070 patent sought to solve problems with, and improve upon, existing database systems. For example, the '070 patent states:

In a variety of fields, data or a set of data, may be represented in a hierarchical fashion. This form of representation may, for example, convey information, such as particular relationships between particular pieces of data and the like. However, manipulating such data representations is not

straight-forward, particularly where the data is arranged in a complex hierarchy. Without loss of generality, one example may include a relational database. Techniques for performing operations on such a database, for example, are computationally complex or otherwise cumbersome. A continuing need, therefore, exists for additional techniques for manipulating data hierarchies.

See '070 Specification at col. 1, ll. 18-29.

35. The invention(s) claimed in the '070 patent solves various technological problems inherent in the then-existing database systems and enables database systems to, among other things, (1) function more efficiently, (2) improve management and/or utilization of disparate data sources, (3) reduce data retrieval latency, (4) improve data processing speeds and/or flexibility, (5) reduce data processing and/or transmission overhead, and (6) improve ease of use.

United States Patent No. 11,194,777

36. On December 7, 2021, the USPTO duly and legally issued United States Patent No. 11,194,777 ("the '777 patent") entitled "Manipulation and/or Analysis of Hierarchical Data" to inventors Karl Schiffmann, Mark Andrews, and Jack J. LeTourneau.

37. The '777 patent is presumed valid under 35 U.S.C. § 282.

38. Lower48 owns all rights, title, and interest in the '777 patent.

39. Lower48 has not granted Shopify a license to the rights under the '777 patent.

40. The '777 patent relates to, among other things, improvements to the functionality of database systems.

41. The invention(s) claimed in the '777 patent is a continuation of the '070 patent and solves the same technological challenges as the '070 patent.

CLAIMS FOR RELIEF

Count I - Direct Infringement of United States Patent No. 7,681,177

42. Lower48, realleges, and incorporates by reference, as if fully set forth here, the allegations of the preceding paragraphs above.

43. On information and belief, Shopify (or those acting on its behalf) makes, uses, sells, imports and/or offers to sell the Shopify GraphQL Products and Services; and makes, uses, sells, sells access to, imports, offers to sell and/or offers to sell access to the Shopify GraphQL System in the United States that infringe (literally and/or under the doctrine of equivalents) at least claim 45 of the '177 patent.

44. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method comprising executing instructions by a processor for transforming between one or more electrical digital signals representing at least a first expression (*e.g.*, a GraphQL query) and one or more electrical digital signals representing at least a second expression (*e.g.*, an Abstract Syntax Tree).



See e.g., <https://graphql.org>.

Available APIs

Admin API

The [Admin API](#) is used to read and write data about merchant stores, products, orders, and more. You can use the Admin API to build [apps](#) that add features to the Shopify admin, the store management interface used by merchants. The Admin API is accessible using either [GraphQL](#) (recommended) or [REST](#).

[Learn more about the Admin API](#)

See e.g., <https://shopify.dev/api/usage>.

GraphQL Admin API

The GraphQL Admin API is a GraphQL-based alternative to the REST-based [Admin API](#), and makes the functionality of the Shopify admin available at a single GraphQL endpoint:

```
POST https://{shop}.myshopify.com/admin/api/2022-07/graphql.json
```

You can access the GraphQL Admin API using the GraphiQL app, curl, or any HTTP client:

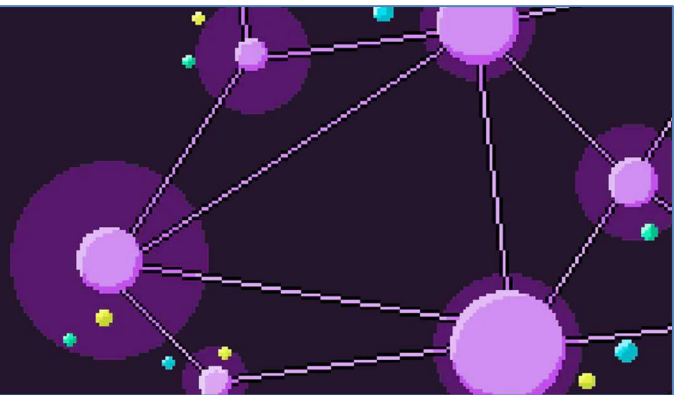
- [Use the GraphiQL app](#)
- [Use curl \(or your preferred HTTP client\)](#)

See e.g., <https://shopify.dev/api/admin/getting-started>.

GraphQL Admin API reference

The Admin API lets you build apps and integrations that extend and enhance the Shopify admin.

This page will help you get up and running with Shopify's GraphQL API.



See e.g., <https://shopify.dev/api/admin-graphql>.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

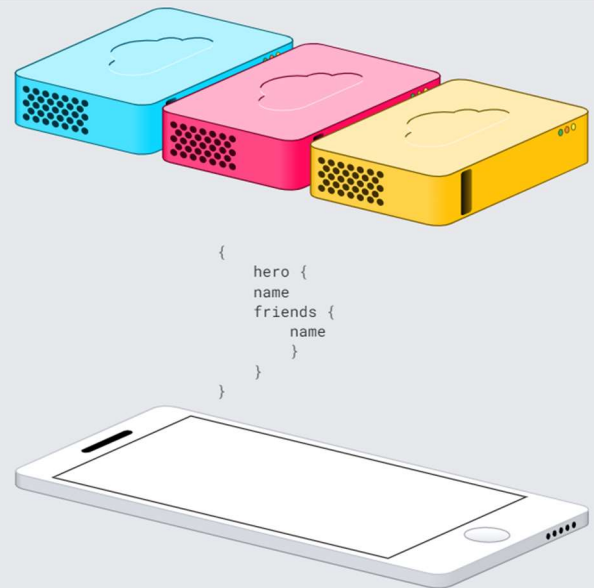
You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

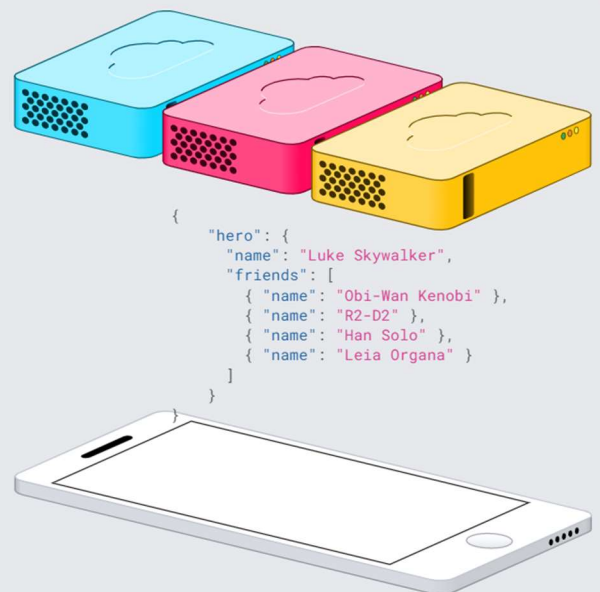
Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



See e.g., <https://graphql.org>.

Execution

After being validated, a GraphQL query is executed by a GraphQL server which returns a result that mirrors the shape of the requested query, typically as JSON.

GraphQL cannot execute a query without a type system, let's use an example type system to illustrate executing a query. This is a part of the same type system used throughout the examples in these articles:

```
type Query {  
  human(id: ID!): Human  
}  
  
type Human {  
  name: String  
  appearsIn: [Episode]  
  starships: [Starship]  
}  
  
enum Episode {  
  NEWHOPE  
  EMPIRE  
  JEDI  
}  
  
type Starship {  
  name: String  
}
```

See e.g., <https://graphql.org/learn/execution/>.

Every GraphQL query goes through three phases. Queries are **parsed**, **validated** and **executed**.

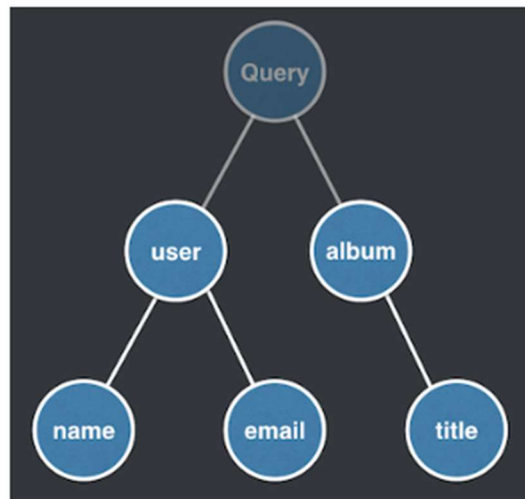
1. **Parse** — A query is **parsed** into an abstract syntax tree (or AST). ASTs are incredibly powerful and behind tools like ESLint, babel, etc. If you want to see what a GraphQL AST looks like, check out astexplorer.net and change JavaScript to GraphQL. You will see a query on the left and an AST on the right.
2. **Validate** — The AST is **validated** against the schema. Checks for correct query syntax and if the fields exist.
3. **Execute** — The runtime **walks** through the AST, starting from the root of the tree, **invokes** resolvers, **collects** up results, and emits JSON.

For this example, we'll refer to this query:

```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

Example

The following example shows how to use `shopifyqlQuery` in the GraphQL Admin API to retrieve the total sales for each month from the start of the year until today.

POST https://{shop_name}.myshopify.com/admin/api/unstable/graphql.json

GraphQL query

```

1  {
2    # "FROM sales SHOW total_sales BY month SINCE -1y UNTIL today" passes a ShopifyQL query to the
   GraphQL query.
3    shopifyqlQuery(query: "FROM sales SHOW total_sales BY month SINCE -1y UNTIL today") {
4      __typename
5      ... on TableResponse {
6        tableData {
7          rowData
8          columns {
9            # Elements in the columns section describe which column properties you want to return.
10             name
11             dataType
12             displayName
13           }
14         }
15       }
16     # parseErrors specifies that you want errors returned, if there were any, and which error
       properties you want to return.
17     parseErrors {
18       code
19       message
20       range {
21         start {
22           line
23           character
24         }
25         end {
26           line
27           character
28         }
29       }
30     }
31   }
32 }

```



```

Response
1  {
2    "data": {
3      "shopifyqlQuery": {
4        "__typename": "TableResponse",
5        "tableData": {
6          "rowData": [
7            [
8              "Feb 2022",
9              "$2,000.00"
10             ],
11             [
12               "Mar 2022",
13               "$3,000.00"
14             ],
15             [
16               "Apr 2022",
17               "$2,500.00"
18             ]
19           ],
20           "columns": [
21             {
22               "name": "month",
23               "dataType": "month",
24               "displayName": "month"
25             },
26             {
27               "name": "total_sales",
28               "dataType": "price",
29               "displayName": "total_sales"
30             }
31           ]
32         },
33         "parseErrors": null
34       }
35     }
36   }

```

See e.g., <https://shopify.dev/api/shopifyql>.

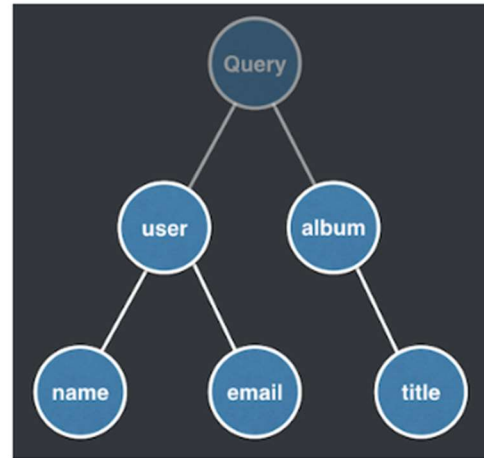
45. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method wherein said expressions have a common view, wherein said common view is other than view one.

For this example, we'll refer to this query:

```
query {
  user {
    name
    email
  }
  album {
    title
  }
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

46. Shopify's direct infringement has damaged Lower48 and caused it to suffer and continue to suffer irreparable harm and damages.

Count II – Direct Infringement of United States Patent No. 10,140,349

47. Lower 48, realleges, and incorporates by reference, as if fully set forth here, the allegations of the preceding paragraphs above.

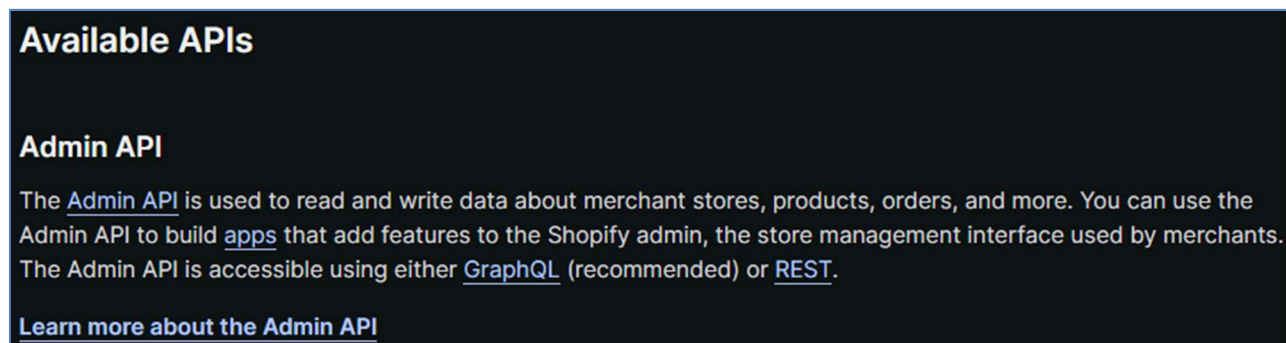
48. On information and belief, Shopify (or those acting on its behalf) makes, uses, sells, imports and/or offers to sell the Shopify GraphQL Products and Services; and makes, uses, sells, sells access to, imports, offers to sell and/or offers to sell access to the Shopify GraphQL System in the United States that infringe (literally and/or under the doctrine of equivalents) at least claim 21 of the '349 patent.

49. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of executing computer

instructions on at least one computing device (*e.g.*, an API server), in which the at least one computing device includes at least one processor and at least one memory, the method comprising executing computer instructions on the at least one processor of the at least one computing device, the computer instructions having been stored on the at least one memory; and storing in the at least one memory of the at least one computing device any results of having executed the computer instructions on the at least one processor of the at least one computing device; wherein the computer instructions to be executed comprise instructions for converting signal values for a first expression (*e.g.*, a GraphQL query) for more convenient processing and/or storage.



See *e.g.*, <https://graphql.org>.



See *e.g.*, <https://shopify.dev/api/usage>.

GraphQL Admin API

The GraphQL Admin API is a GraphQL-based alternative to the REST-based [Admin API](#), and makes the functionality of the Shopify admin available at a single GraphQL endpoint:

POST `https://{shop}.myshopify.com/admin/api/2022-07/graphql.json`

You can access the GraphQL Admin API using the GraphiQL app, curl, or any HTTP client:

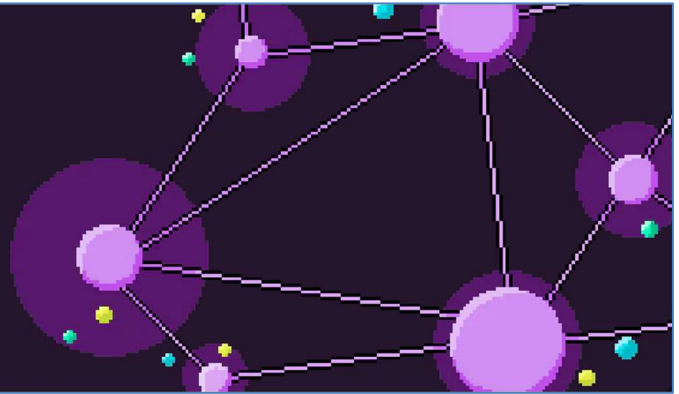
- [Use the GraphiQL app](#)
- [Use curl \(or your preferred HTTP client\)](#)

See e.g., <https://shopify.dev/api/admin/getting-started>.

GraphQL Admin API reference

The Admin API lets you build apps and integrations that extend and enhance the Shopify admin.

This page will help you get up and running with Shopify's GraphQL API.



See e.g., <https://shopify.dev/api/admin-graphql>.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

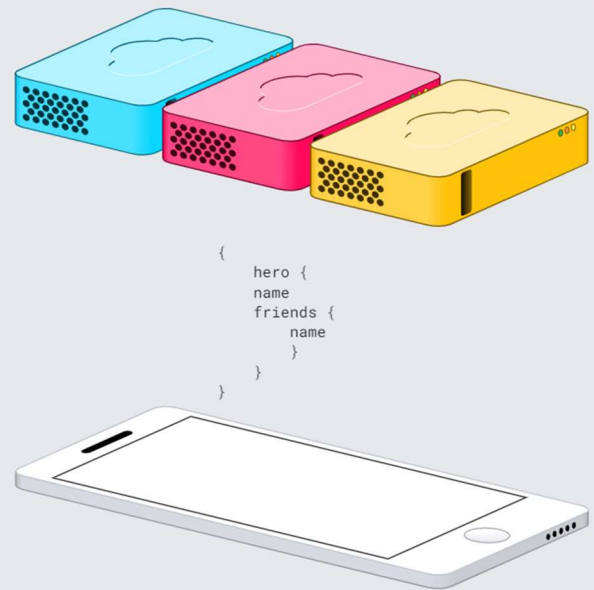
You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

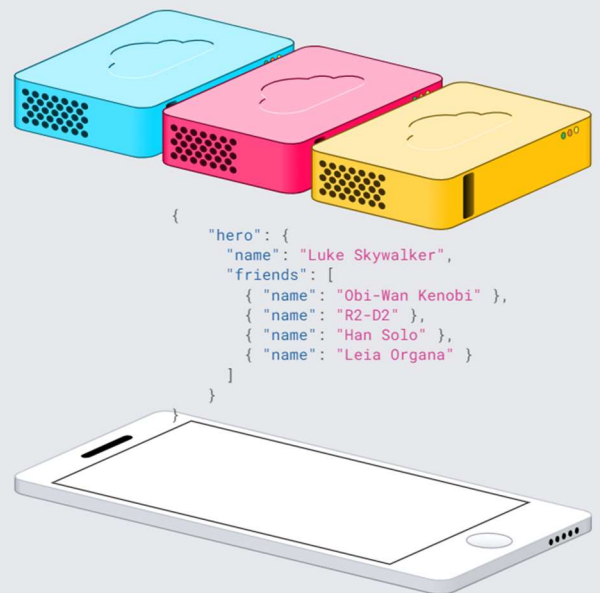
Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



See e.g., <https://graphql.org>.

Execution

After being validated, a GraphQL query is executed by a GraphQL server which returns a result that mirrors the shape of the requested query, typically as JSON.

GraphQL cannot execute a query without a type system, let's use an example type system to illustrate executing a query. This is a part of the same type system used throughout the examples in these articles:

```
type Query {
  human(id: ID!): Human
}

type Human {
  name: String
  appearsIn: [Episode]
  starships: [Starship]
}

enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}

type Starship {
  name: String
}
```

See e.g., <https://graphql.org/learn/execution/>.

Use the GraphiQL app

We recommend installing [Shopify's own GraphiQL app](#) to explore your shop's data using the GraphQL Admin API. After you've installed the app, you can test it by running the following query:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```
1 query {
2   shop {
3     name
4     primaryDomain {
5       url
6       host
7     }
8   }
9 }
```

 JSON response

```
1 {
2   "data": {
3     "shop": {
4       "name": "graphql",
5       "primaryDomain": {
6         "url": "https://graphql.myshopify.com",
7         "host": "graphql.myshopify.com"
8       }
9     }
10  }
11 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

Example

The following example shows how to use [shopifyqlQuery](#) in the GraphQL Admin API to retrieve the total sales for each month from the start of the year until today.


```

POST https://{shop_name}.myshopify.com/admin/api/unstable/graphql.json

GraphQL query

1  {
2    # "FROM sales SHOW total_sales BY month SINCE -1y UNTIL today" passes a ShopifyQL query to the
   GraphQL query.
3    shopifyqlQuery(query: "FROM sales SHOW total_sales BY month SINCE -1y UNTIL today") {
4      __typename
5      ... on TableResponse {
6        tableData {
7          rowData
8          columns {
9            # Elements in the columns section describe which column properties you want to return.
10             name
11             dataType
12             displayName
13           }
14         }
15       }
16       # parseErrors specifies that you want errors returned, if there were any, and which error
   properties you want to return.
17       parseErrors {
18         code
19         message
20         range {
21           start {
22             line
23             character
24           }
25           end {
26             line
27             character
28           }
29         }
30       }
31     }
32   }

```

See e.g., <https://shopify.dev/api/shopifyql>.

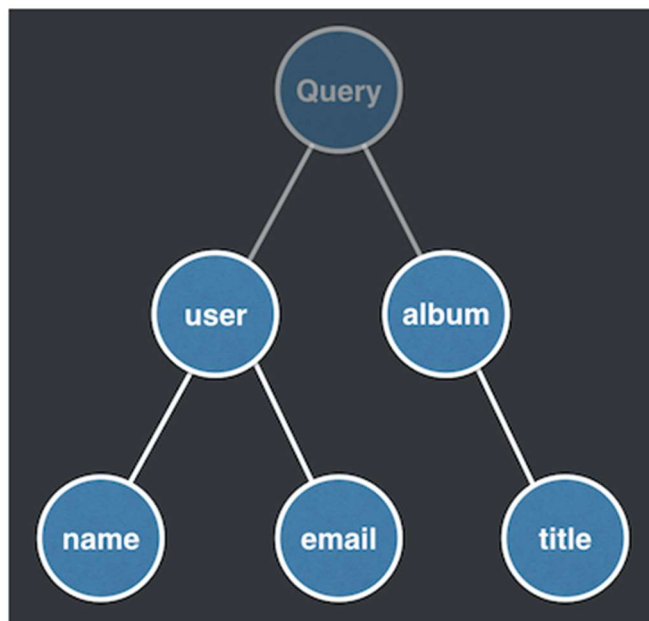
GraphQL is a specification typically used for remote client-server communications. Unlike SQL, GraphQL is agnostic to the data source(s) used to retrieve data and persist changes. Accessing and manipulating data is performed with arbitrary functions called **resolvers**. GraphQL coordinates and aggregates the data from these resolver functions, then returns the result to the client. Generally, these resolver functions should delegate to a **business logic layer** responsible for communicating with the various underlying data sources. These data sources could be remote APIs, databases, **local cache**, and nearly anything else your programming language can access.

See e.g., <https://graphql.org/faq/#is-graphql-a-database-language-like-sql>


```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.

See e.g., <https://graphql.org/>.

50. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of executing computer instructions wherein executing the converting instructions further comprises accessing, from memory, signal values for a first expression, (e.g. a GraphQL query).

Every GraphQL query goes through three phases. Queries are **parsed**, **validated** and **executed**.

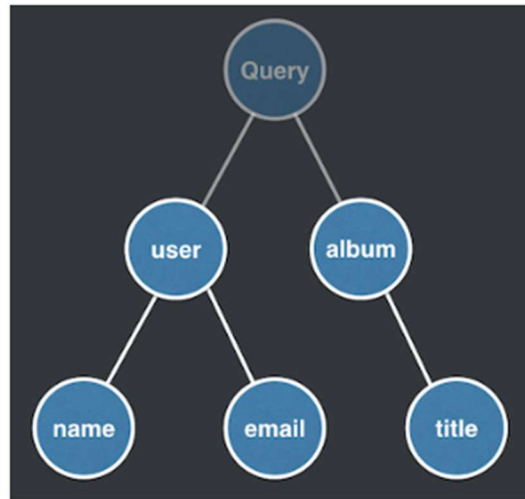
1. **Parse** — A query is **parsed** into an abstract syntax tree (or AST). ASTs are incredibly powerful and behind tools like ESLint, babel, etc. If you want to see what a GraphQL AST looks like, check out astexplorer.net and change JavaScript to GraphQL. You will see a query on the left and an AST on the right.
2. **Validate** — The AST is **validated** against the schema. Checks for correct query syntax and if the fields exist.
3. **Execute** — The runtime **walks** through the AST, starting from the root of the tree, **invokes** resolvers, **collects** up results, and emits JSON.

For this example, we'll refer to this query:

```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

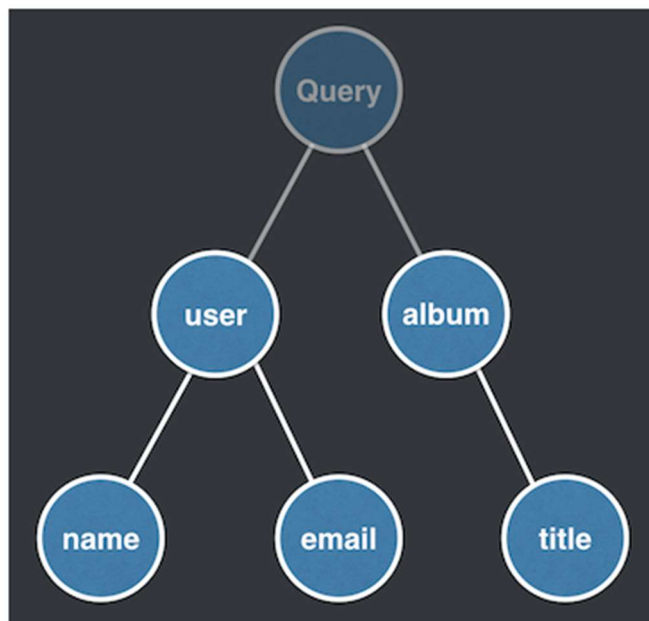
See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

51. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of executing computer instructions wherein executing the converting instructions further comprises transforming the signal values for the first expression (e.g. a GraphQL query) to signal values for a second expression, (e.g. an Abstract Syntax Tree), the first expression having a first expression type and the second expression having a second expression type.

```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

52. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of executing computer instructions wherein the first and second expression types comprise at least one of the following expression types: a hierarchical edge and/or node labeled tree or a symbol string.

GraphQL operations are hierarchical and composed, describing a tree of information. While Scalar types describe the leaf values of these hierarchical operations, Objects describe the intermediate levels.

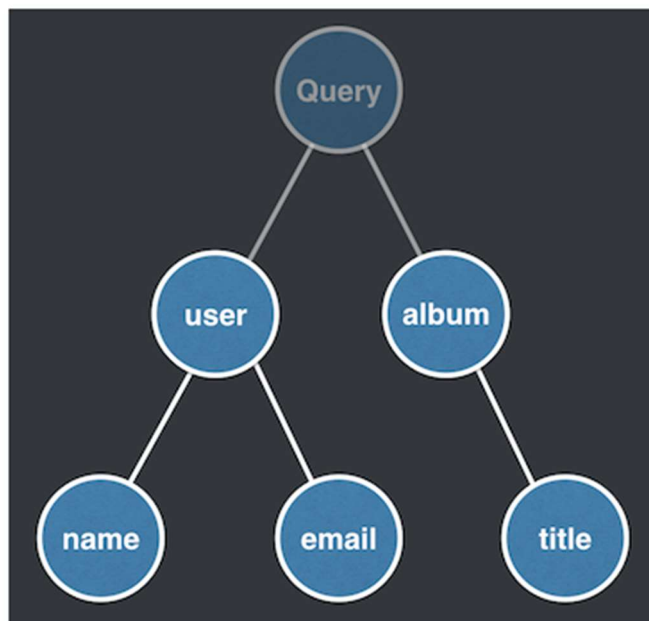
GraphQL Objects represent a list of named fields, each of which yield a value of a specific type. Object values should be serialized as ordered maps, where the selected field names (or aliases) are the keys and the result of evaluating the field is the value, ordered by the order in which they appear in the selection set.

See e.g., <http://spec.graphql.org/draft/>.

```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

53. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of executing computer instructions wherein executing the converting instructions further comprises storing, in the memory, the signal values corresponding to the second expression, (e.g., an Abstract Syntax Tree).

Every GraphQL query goes through three phases. Queries are **parsed**, **validated** and **executed**.

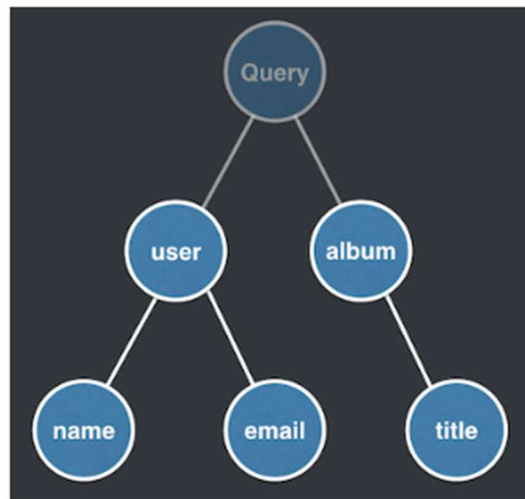
1. **Parse** — A query is **parsed** into an abstract syntax tree (or AST). ASTs are incredibly powerful and behind tools like ESLint, babel, etc. If you want to see what a GraphQL AST looks like, check out astexplorer.net and change JavaScript to GraphQL. You will see a query on the left and an AST on the right.
2. **Validate** — The AST is **validated** against the schema. Checks for correct query syntax and if the fields exist.
3. **Execute** — The runtime **walks** through the AST, starting from the root of the tree, **invokes** resolvers, **collects** up results, and emits JSON.

For this example, we'll refer to this query:

```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

When this query is **parsed**, it's converted to an AST, or a tree.



Query represented as a tree

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

Available APIs

Admin API

The [Admin API](#) is used to read and write data about merchant stores, products, orders, and more. You can use the Admin API to build [apps](#) that add features to the Shopify admin, the store management interface used by merchants. The Admin API is accessible using either [GraphQL](#) (recommended) or [REST](#).

[Learn more about the Admin API](#)

See e.g., <https://shopify.dev/api/usage>.

54. Shopify's direct infringement has damaged Lower48 and caused it to suffer and continue to suffer irreparable harm and damages.

Count III – Direct Infringement of United States Patent No. 11,100,070

55. Lower 48, realleges, and incorporates by reference, as if fully set forth here, the allegations of the preceding paragraphs above.

56. On information and belief, Shopify (or those acting on its behalf) makes, uses, sells, imports and/or offers to sell the Shopify GraphQL Products and Services; and makes, uses, sells, sells access to, imports, offers to sell and/or offers to sell access to the Shopify GraphQL System in the United States that infringe (literally and/or under the doctrine of equivalents) at least claim 1 of the '070 patent.

57. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof.



See e.g., <https://graphql.org>.

Available APIs

Admin API

The [Admin API](#) is used to read and write data about merchant stores, products, orders, and more. You can use the Admin API to build [apps](#) that add features to the Shopify admin, the store management interface used by merchants. The Admin API is accessible using either [GraphQL](#) (recommended) or [REST](#).

[Learn more about the Admin API](#)

See e.g., <https://shopify.dev/api/usage>.

GraphQL Admin API

The GraphQL Admin API is a GraphQL-based alternative to the REST-based [Admin API](#), and makes the functionality of the Shopify admin available at a single GraphQL endpoint:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

You can access the GraphQL Admin API using the GraphiQL app, curl, or any HTTP client:

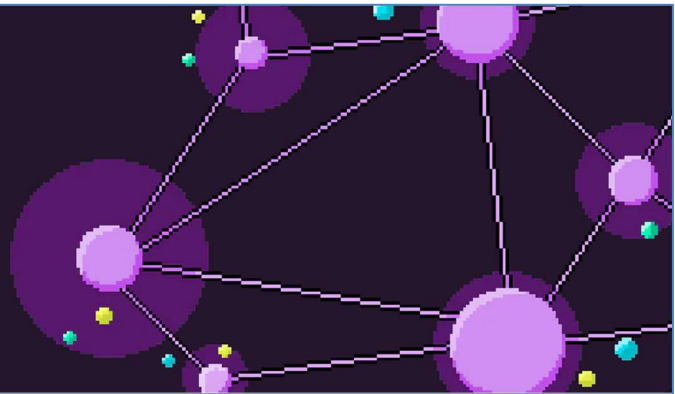
- [Use the GraphiQL app](#)
- [Use curl \(or your preferred HTTP client\)](#)

See e.g., <https://shopify.dev/api/admin/getting-started>.

GraphQL Admin API reference

The Admin API lets you build apps and integrations that extend and enhance the Shopify admin.

This page will help you get up and running with Shopify's GraphQL API.



See e.g., <https://shopify.dev/api/admin-graphql>.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

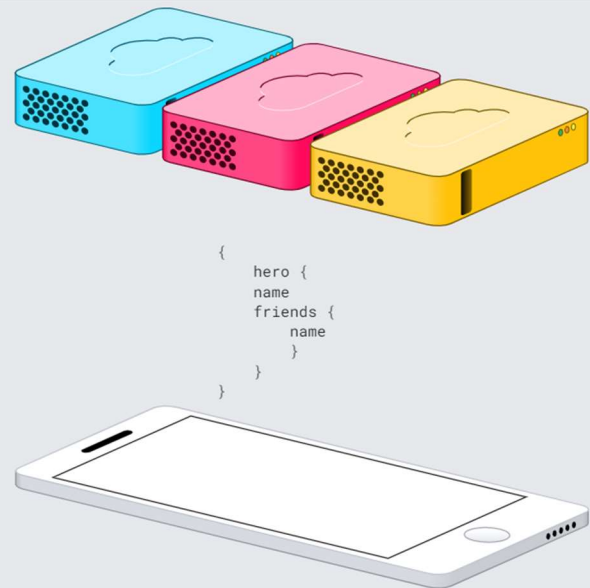
You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

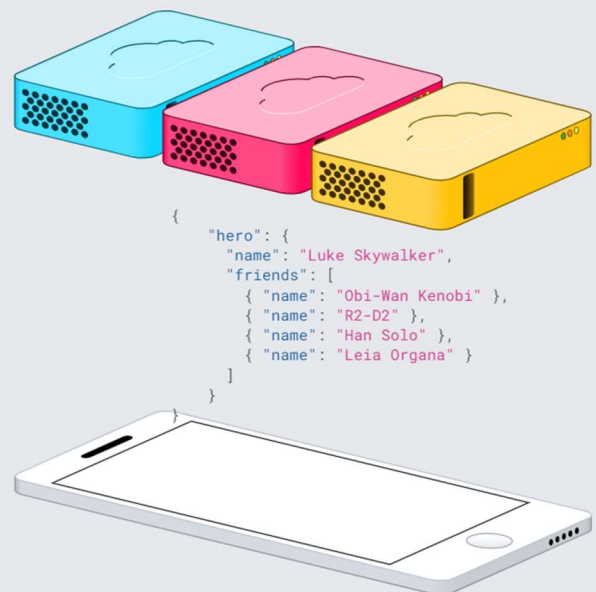
Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



See e.g., <https://graphql.org>.

Execution

After being validated, a GraphQL query is executed by a GraphQL server which returns a result that mirrors the shape of the requested query, typically as JSON.

GraphQL cannot execute a query without a type system, let's use an example type system to illustrate executing a query. This is a part of the same type system used throughout the examples in these articles:

```
type Query {  
  human(id: ID!): Human  
}  
  
type Human {  
  name: String  
  appearsIn: [Episode]  
  starships: [Starship]  
}  
  
enum Episode {  
  NEWHOPE  
  EMPIRE  
  JEDI  
}  
  
type Starship {  
  name: String  
}
```

See e.g., <https://graphql.org/learn/execution/>.

Use the GraphQL app

We recommend installing [Shopify's own GraphQL app](#) to explore your shop's data using the GraphQL Admin API. After you've installed the app, you can test it by running the following query:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1 query {
2   shop {
3     name
4     primaryDomain {
5       url
6       host
7     }
8   }
9 }
```

JSON response

```

1 {
2   "data": {
3     "shop": {
4       "name": "graphql",
5       "primaryDomain": {
6         "url": "https://graphql.myshopify.com",
7         "host": "graphql.myshopify.com"
8       }
9     }
10  }
11 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

Example

The following example shows how to use [shopifyqlQuery](#) in the GraphQL Admin API to retrieve the total sales for each month from the start of the year until today.

```

POST https://{shop_name}.myshopify.com/admin/api/unstable/graphql.json

GraphQL query

1  {
2    # "FROM sales SHOW total_sales BY month SINCE -1y UNTIL today" passes a ShopifyQL query to the
   GraphQL query.
3    shopifyqlQuery(query: "FROM sales SHOW total_sales BY month SINCE -1y UNTIL today") {
4      __typename
5      ... on TableResponse {
6        tableData {
7          rowData
8          columns {
9            # Elements in the columns section describe which column properties you want to return.
10             name
11             dataType
12             displayName
13           }
14         }
15       }
16     # parseErrors specifies that you want errors returned, if there were any, and which error
       properties you want to return.
17     parseErrors {
18       code
19       message
20       range {
21         start {
22           line
23           character
24         }
25         end {
26           line
27           character
28         }
29       }
30     }
31   }
32 }

```

See e.g., <https://shopify.dev/api/shopifyql>.

GraphQL is a specification typically used for remote client-server communications. Unlike SQL, GraphQL is agnostic to the data source(s) used to retrieve data and persist changes. Accessing and manipulating data is performed with arbitrary functions called **resolvers**. GraphQL coordinates and aggregates the data from these resolver functions, then returns the result to the client. Generally, these resolver functions should delegate to a **business logic layer** responsible for communicating with the various underlying data sources. These data sources could be remote APIs, databases, **local cache**, and nearly anything else your programming language can access.

See e.g., <https://graphql.org/faq/#is-graphql-a-database-language-like-sql>

58. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof, comprising accessing instructions from one or more physical memory

devices for execution by one or more processors; executing the instructions accessed from the one or more physical memory devices by the one or more processors, and storing, in at least one of the physical memory devices, signal values, including numerical signal values, resulting from having executed the accessed instructions on the one or more processors, wherein the one or more physical memory devices also store the database, or the portion thereof.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

Available APIs

Admin API

The [Admin API](#) is used to read and write data about merchant stores, products, orders, and more. You can use the Admin API to build [apps](#) that add features to the Shopify admin, the store management interface used by merchants. The Admin API is accessible using either [GraphQL](#) (recommended) or [REST](#).

[Learn more about the Admin API](#)

See e.g., <https://shopify.dev/api/usage>.

GraphQL Admin API

The GraphQL Admin API is a GraphQL-based alternative to the REST-based [Admin API](#), and makes the functionality of the Shopify admin available at a single GraphQL endpoint:

```
POST https://{shop}.myshopify.com/admin/api/2022-07/graphql.json
```

You can access the GraphQL Admin API using the GraphiQL app, curl, or any HTTP client:

- [Use the GraphiQL app](#)
- [Use curl \(or your preferred HTTP client\)](#)

See e.g., <https://shopify.dev/api/admin/getting-started>.

Use the GraphQL app

We recommend installing [Shopify's own GraphQL app](#) to explore your shop's data using the GraphQL Admin API. After you've installed the app, you can test it by running the following query:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1 query {
2   shop {
3     name
4     primaryDomain {
5       url
6       host
7     }
8   }
9 }
```

JSON response

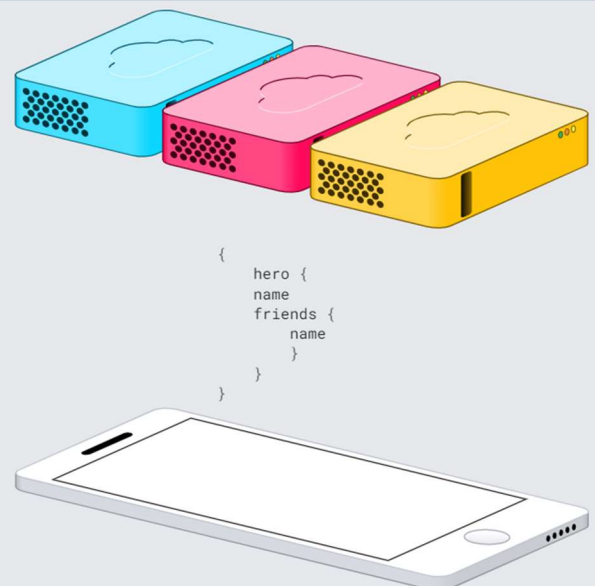
```

1 {
2   "data": {
3     "shop": {
4       "name": "graphql",
5       "primaryDomain": {
6         "url": "https://graphql.myshopify.com",
7         "host": "graphql.myshopify.com"
8       }
9     }
10  }
11 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

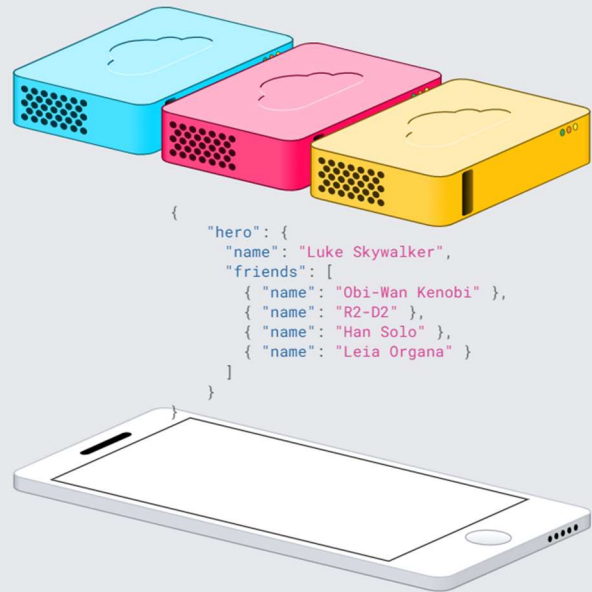
Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



See e.g., <https://graphql.org>.

Execution

After being validated, a GraphQL query is executed by a GraphQL server which returns a result that mirrors the shape of the requested query, typically as JSON.

GraphQL cannot execute a query without a type system, let's use an example type system to illustrate executing a query. This is a part of the same type system used throughout the examples in these articles:

```

type Query {
  human(id: ID!): Human
}

type Human {
  name: String
  appearsIn: [Episode]
  starships: [Starship]
}

enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}

type Starship {
  name: String
}

```

See e.g., <https://graphql.org/learn/execution/>.

Example query

In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under QueryRoot and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11  }
12  }
```

JSON response

```

1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           }
12         ]
13       }
14     }
15   }
16 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

59. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof wherein the accessed instructions to transform the database, or the portion thereof, to the form of a hierarchically structured tree, (e.g., a GraphQL response)

via one or more numerical signal values corresponding to content within the database, or the portion thereof.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

Every GraphQL query goes through three phases. Queries are **parsed**, **validated** and **executed**.

1. **Parse** — A query is **parsed** into an abstract syntax tree (or AST). ASTs are incredibly powerful and behind tools like ESLint, babel, etc. If you want to see what a GraphQL AST looks like, check out astexplorer.net and change JavaScript to GraphQL. You will see a query on the left and an AST on the right.
2. **Validate** — The AST is **validated** against the schema. Checks for correct query syntax and if the fields exist.
3. **Execute** — The runtime **walks** through the AST, starting from the root of the tree, **invokes** resolvers, **collects** up results, and emits JSON.

For this example, we'll refer to this query:

```
query {  
  user {  
    name  
    email  
  }  
  album {  
    title  
  }  
}
```

Query for later reference

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

GraphQL operations are hierarchical and composed, describing a tree of information. While Scalar types describe the leaf values of these hierarchical operations, Objects describe the intermediate levels.

GraphQL Objects represent a list of named fields, each of which yield a value of a specific type. Object values should be serialized as ordered maps, where the selected field names (or aliases) are the keys and the result of evaluating the field is the value, ordered by the order in which they appear in the selection set.

See e.g., <https://spec.graphQL.org/draft>.

3.5 Scalars

ScalarTypeDefinition :

*Description*_{opt} **scalar** *Name* *Directives*_{[Const]opt}

Scalar types represent primitive leaf values in a GraphQL type system. GraphQL responses take the form of a hierarchical tree; the leaves on these trees are GraphQL scalars.

Int

The Int scalar type represents a signed 32-bit numeric non-fractional value. Response formats that support a 32-bit integer or a number type should use that type to represent this scalar.

See e.g., <https://spec.graphql.org/June2018/#sec-Scalars>.

Example query

In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under [QueryRoot](#) and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11   }
12 }
```

```

JSON response
1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11          ]
12        }
13      }
14    }
15  }
16 }

```

Example query

In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under [QueryRoot](#) and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11   }
12 }

```

See e.g., <https://shopify.dev/api/admin/getting-started>.

60. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof wherein the executing the transformation instructions comprises generating the hierarchically structured tree via the one or more numerical signal values,

the hierarchically structured tree comprising electronic content including binary digital signals and/or states.

3.5 Scalars

ScalarTypeDefinition :

*Description*_{opt} **scalar** *Name Directives*_{[Const]opt}

Scalar types represent primitive leaf values in a GraphQL type system. GraphQL responses take the form of a hierarchical tree; the leaves on these trees are GraphQL scalars.

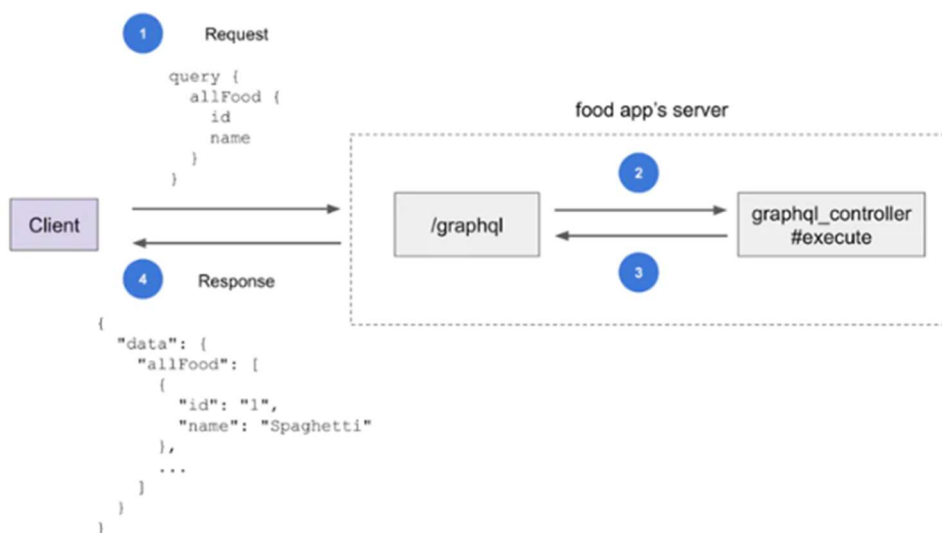
Int

The Int scalar type represents a signed 32-bit numeric non-fractional value. Response formats that support a 32-bit integer or a number type should use that type to represent this scalar.

See e.g., <https://spec.graphql.org/June2018/#sec-Scalars>.

What's Happening Behind the Scenes?

Recall from part one, GraphQL has this single "smart" endpoint that bundles all different types of RESTful actions under one endpoint. This is what happens when you make a request and get a response back.



When you execute the query:

- 1 You call the graphql endpoint with your request (for example, query and variables).
- 2 The graphql endpoint then calls the execute method from the graphql_controller to process your request.
- 3 The method renders a JSON containing a response catered to your request.
- 4 You get a response back.

See e.g., <https://shopify.engineering/understanding-graphql-beginner-part-two>.

```

1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           }
12         ]
13       }
14     }
15   }
16 }

```

See e.g., <https://shopify.dev/api/admin/getting-started>.

Scalar types

Scalar types are similar to primitive types in your favorite programming language. They always resolve to concrete data.

GraphQL's default scalar types are:

- **Int**: A signed 32-bit integer
- **Float**: A signed double-precision floating-point value
- **String**: A UTF-8 character sequence
- **Boolean**: `true` or `false`

See e.g., <https://www.apollographql.com/docs/apollo-server/schema/schema/>.

POST https://{shop}.myshopify.com/admin/api/2022-07/graphql.json

Terminal

```

$ curl -X POST \
$ https://{shop}.myshopify.com/admin/api/2022-07/graphql.json \
$ -H 'Content-Type: application/graphql' \
$ -H 'X-Shopify-Access-Token: {password}' \
$ -d '
$ {
$   products(first: 5) {
$     edges {
$       node {
$         id
$         handle
$       }
$     }
$     pageInfo {
$       hasNextPage
$     }
$   }
$ }
$ '

```

Hide response

JSON response

```

1  {
2    "data": {
3      "products": {
4        "edges": [
5          {
6            "node": {
7              "id": "gid://shopify/Product/1974208299030",
8              "handle": "rough-snowflake-camisole"
9            }
10         ]
11       },
12       "pageInfo": {
13         "hasNextPage": true
14       }
15     }
16   }
17 }

```

See e.g., <https://www.apollographql.com/docs/apollo-server/schema/schema/>.

```

26     "node": {
27         "id":
28         "gid://shopify/DiscountAutomaticNode/299501151"
29         ,
30         "automaticDiscount": {
31             "title": "My automatic discount",
32             "summary": "$100.00 off all products •
33             Minimum quantity of 1",
34             "customerGets": {
35                 "items": {
36                     "allItems": true
37                 }
38             },
39             "minimumRequirement": {
40                 "greaterThanOrEqualToQuantity": "1"
41             }
42         }
43     }
44 }

```

See e.g., [https://shopify.dev/api/admin-graphql/2022-](https://shopify.dev/api/admin-graphql/2022-01/queries/automaticDiscountNodes#argument-automaticdiscountnodes-reverse)

01/queries/automaticDiscountNodes#argument-automaticdiscountnodes-reverse.

61. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof wherein the executing the transformation instructions comprises presenting a query to the database, or the portion thereof, via at least one numerical signal value to fetch the content within the database, or the portion thereof.

Example query

In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under [QueryRoot](#) and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11   }
12 }
```

JSON response

```

1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           ]
12         }
13       }
14     }
15   }
16 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

62. Shopify's direct infringement has damaged Lower48 and caused it to suffer and continue to suffer irreparable harm and damages.

Count IV – Direct Infringement of United States Patent No. 11,194,777

63. Lower 48, realleges, and incorporates by reference, as if fully set forth here, the allegations of the preceding paragraphs above.

64. On information and belief, Shopify (or those acting on its behalf) makes, uses, sells, imports and/or offers to sell the Shopify GraphQL Products and Services; and makes, uses, sells, sells access to, imports, offers to sell and/or offers to sell access to the Shopify GraphQL System in the United States that infringe (literally and/or under the doctrine of equivalents) at least claim 1 of the '777 patent.

65. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof, comprising accessing instructions from one or more physical memory devices for execution by one or more processors, executing the instructions accessed from the one or more physical memory devices by the one or more processors, and storing, in at least one of the physical memory devices, signal values, including numerical signal values, resulting from having executed the accessed instructions on the one or more processors, wherein the one or more physical memory devices also store the database or the portion thereof.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

Available APIs

Admin API

The [Admin API](#) is used to read and write data about merchant stores, products, orders, and more. You can use the Admin API to build [apps](#) that add features to the Shopify admin, the store management interface used by merchants. The Admin API is accessible using either [GraphQL](#) (recommended) or [REST](#).

[Learn more about the Admin API](#)

See e.g., <https://shopify.dev/api/usage>.

GraphQL Admin API

The GraphQL Admin API is a GraphQL-based alternative to the REST-based [Admin API](#), and makes the functionality of the Shopify admin available at a single GraphQL endpoint:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

You can access the GraphQL Admin API using the GraphiQL app, curl, or any HTTP client:

- [Use the GraphiQL app](#)
- [Use curl \(or your preferred HTTP client\)](#)

See e.g., <https://shopify.dev/api/admin/getting-started>.

Use the GraphiQL app

We recommend installing [Shopify's own GraphiQL app](#) [\[icon\]](#) to explore your shop's data using the GraphQL Admin API. After you've installed the app, you can test it by running the following query:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  query {
2    shop {
3      name
4      primaryDomain {
5        url
6        host
7      }
8    }
9  }
```

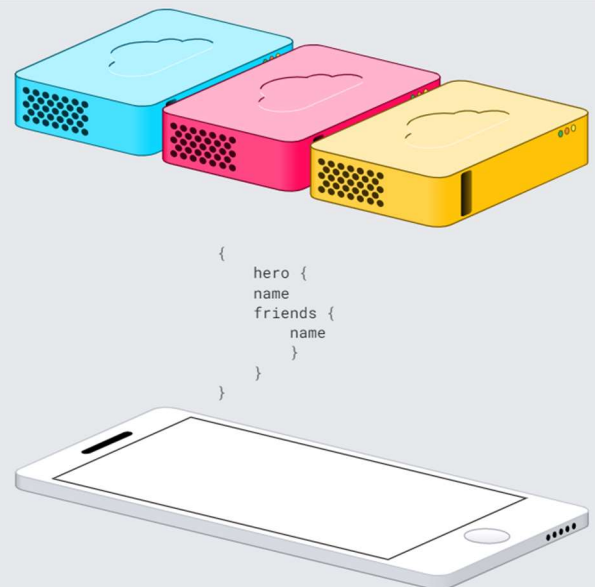


```
JSON response
1 {
2   "data": {
3     "shop": {
4       "name": "graphql",
5       "primaryDomain": {
6         "url": "https://graphql.myshopify.com",
7         "host": "graphql.myshopify.com"
8       }
9     }
10  }
11 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

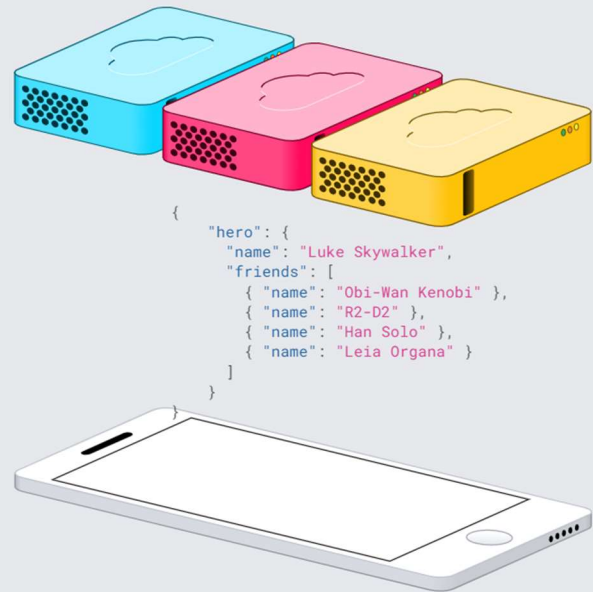
Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



Get many resources in a single request

GraphQL queries access not just the properties of one resource but also smoothly follow references between them. While typical REST APIs require loading from multiple URLs, GraphQL APIs get all the data your app needs in a single request. Apps using GraphQL can be quick even on slow mobile network connections.



See e.g., <https://graphql.org>.

Execution

After being validated, a GraphQL query is executed by a GraphQL server which returns a result that mirrors the shape of the requested query, typically as JSON.

GraphQL cannot execute a query without a type system, let's use an example type system to illustrate executing a query. This is a part of the same type system used throughout the examples in these articles:

```

type Query {
  human(id: ID!): Human
}

type Human {
  name: String
  appearsIn: [Episode]
  starships: [Starship]
}

enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}

type Starship {
  name: String
}

```

See e.g., <https://graphql.org/learn/execution/>.

Example query

In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under QueryRoot and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11  }
12 }
```

JSON response

```

1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           }
12         ]
13       }
14     }
15   }
16 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

66. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof, wherein the accessed instructions transform the database, or the portion thereof, to the form of a hierarchically structured tree (e.g. a GraphQL response) via one

or more numerical signal values corresponding to electronic content within the database, or the portion thereof.

Executing queries

To better understand resolvers, you need to know how queries are executed.

Every GraphQL query goes through three phases. Queries are **parsed**, **validated** and **executed**.

1. **Parse** — A query is **parsed** into an abstract syntax tree (or AST). ASTs are incredibly powerful and behind tools like ESLint, babel, etc. If you want to see what a GraphQL AST looks like, check out astexplorer.net and change JavaScript to GraphQL. You will see a query on the left and an AST on the right.
2. **Validate** — The AST is **validated** against the schema. Checks for correct query syntax and if the fields exist.
3. **Execute** — The runtime **walks** through the AST, starting from the root of the tree, **invokes** resolvers, **collects** up results, and emits JSON.

See e.g., <https://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55>.

GraphQL operations are hierarchical and composed, describing a tree of information. While Scalar types describe the leaf values of these hierarchical operations, Objects describe the intermediate levels.

GraphQL Objects represent a list of named fields, each of which yield a value of a specific type. Object values should be serialized as ordered maps, where the selected field names (or aliases) are the keys and the result of evaluating the field is the value, ordered by the order in which they appear in the selection set.

See e.g., <http://spec.graphql.org/draft/>.

3.5 Scalars

ScalarTypeDefinition :

*Description*_{opt} **scalar** *Name* *Directives*_{[Const]opt}

Scalar types represent primitive leaf values in a GraphQL type system. GraphQL responses take the form of a hierarchical tree; the leaves on these trees are GraphQL scalars.

Int

The Int scalar type represents a signed 32-bit numeric non-fractional value. Response formats that support a 32-bit integer or a number type should use that type to represent this scalar.

See e.g., <https://spec.graphql.org/June2018/#sec-Scalars>.

Example query

In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under [QueryRoot](#) and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11   }
12 }
```

JSON response

```

1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           ]
12         }
13       }
14     }
15   }
16 }
```

See e.g., <https://shopify.dev/api/admin/getting-started>.

67. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof, wherein the executing the transformation instructions comprises generating the hierarchically structured tree, (e.g. a GraphQL response) via the one or more numerical signal values, wherein the hierarchically structured tree comprises the electronic content including binary digital signals and/or states.

3.5 Scalars

ScalarTypeDefinition :

*Description_{opt} **scalar** Name Directives_{[Const]opt}*

Scalar types represent primitive leaf values in a GraphQL type system. GraphQL responses take the form of a hierarchical tree; the leaves on these trees are GraphQL scalars.

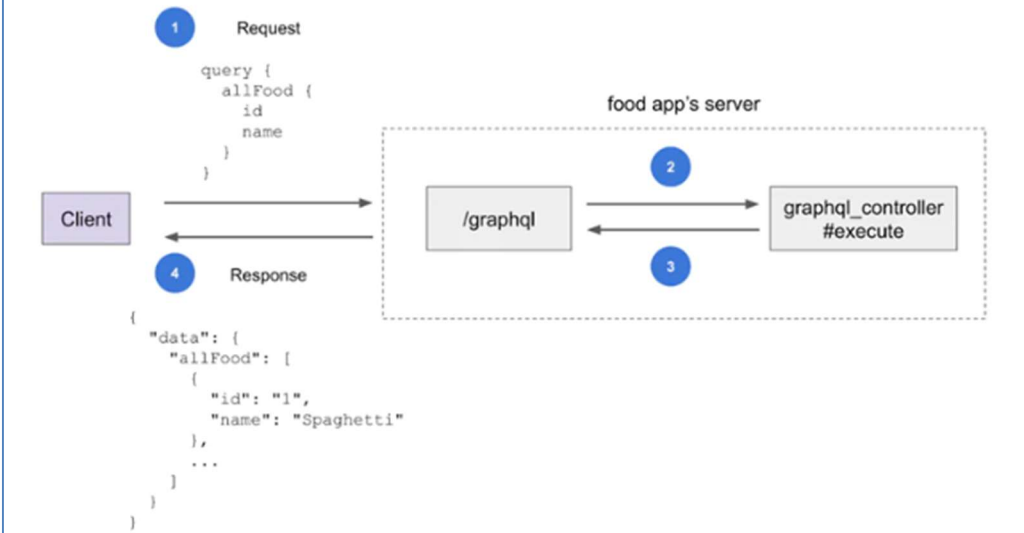
Int

The Int scalar type represents a signed 32-bit numeric non-fractional value. Response formats that support a 32-bit integer or a number type should use that type to represent this scalar.

See e.g., <https://spec.graphql.org/June2018/#sec-Scalars>.

What's Happening Behind the Scenes?

Recall from part one, GraphQL has this single "smart" endpoint that bundles all different types of RESTful actions under one endpoint. This is what happens when you make a request and get a response back.



When you execute the query:

- 1 You call the graphql endpoint with your request (for example, query and variables).
- 2 The graphql endpoint then calls the execute method from the graphql_controller to process your request.
- 3 The method renders a JSON containing a response catered to your request.
- 4 You get a response back.

See e.g., <https://shopify.engineering/understanding-graphql-beginner-part-two>.

```

JSON response
1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           ]
12         }
13       }
14     }
15   }
16 }

```

See e.g., <https://shopify.dev/api/admin/getting-started>.

Scalar types

Scalar types are similar to primitive types in your favorite programming language. They always resolve to concrete data.

GraphQL's default scalar types are:

- `Int`: A signed 32-bit integer
- `Float`: A signed double-precision floating-point value
- `String`: A UTF-8 character sequence
- `Boolean`: `true` or `false`

See e.g., <https://www.apollographql.com/docs/apollo-server/schema/schema/>.


```

26     "node": {
27         "id":
28           "gid://shopify/DiscountAutomaticNode/299501151"
29       ,
30       "automaticDiscount": {
31         "title": "My automatic discount",
32         "summary": "$100.00 off all products •
33           Minimum quantity of 1",
34         "customerGets": {
35           "items": {
36             "allItems": true
37           }
38         },
39         "minimumRequirement": {
40           "greaterThanOrEqualToQuantity": "1"
41         }
42       }
43     }
44   }

```

See e.g., [https://shopify.dev/api/admin-graphql/2022-](https://shopify.dev/api/admin-graphql/2022-01/queries/automaticDiscountNodes#argument-automaticdiscountnodes-reverse)

[01/queries/automaticDiscountNodes#argument-automaticdiscountnodes-reverse](https://shopify.dev/api/admin-graphql/2022-01/queries/automaticDiscountNodes#argument-automaticdiscountnodes-reverse).

68. On information and belief, the Shopify GraphQL Products and Services and Shopify GraphQL System employ and provide a method of querying a database, or a portion thereof, wherein the executing the transformation instructions comprises presenting a GraphQL query to the database, or the portion thereof, via one or more probe numerical signal values to fetch the electronic content corresponding to the one or more numerical signal values.

ShopifyQL overview

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

See e.g., <https://shopify.dev/api/shopifyql>.

GraphQL Admin API

The GraphQL Admin API is a GraphQL-based alternative to the REST-based [Admin API](#), and makes the functionality of the Shopify admin available at a single GraphQL endpoint:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

You can access the GraphQL Admin API using the GraphiQL app, curl, or any HTTP client:

- [Use the GraphiQL app](#)
- [Use curl \(or your preferred HTTP client\)](#)

See e.g., <https://shopify.dev/api/admin/getting-started>.

Example query


In GraphQL, queries are the equivalent of REST's GET action verb. They generally begin with one of the objects listed under [QueryRoot](#) and can get data from any connections that object has. Even though a POST is being sent to the GraphQL endpoint, if the body only contains queries, then data will only be retrieved and not modified.

The following example shows a query for the quantity of inventory items available at a location:

POST <https://{shop}.myshopify.com/admin/api/2022-07/graphql.json>

```

1  {
2    inventoryItem(id: "gid://shopify/InventoryItem/19848949301270") {
3      id
4      inventoryLevels(first: 10) {
5        edges {
6          node {
7            available
8          }
9        }
10     }
11   }
12 }
```



```

1  {
2    "data": {
3      "inventoryItem": {
4        "id": "gid://shopify/InventoryItem/19848949301270",
5        "inventoryLevels": {
6          "edges": [
7            {
8              "node": {
9                "available": 5
10             }
11           ]
12         }
13       }
14     }
15   }
16 }

```

See e.g., <https://shopify.dev/api/admin/getting-started>.

69. Shopify's direct infringement has damaged Lower48 and caused it to suffer and continue to suffer irreparable harm and damages.

JURY DEMANDED

70. Pursuant to Federal Rule of Civil Procedure 38(b), Lower48 hereby requests a trial by jury on all issues so triable.

PRAYER FOR RELIEF

Lower48 respectfully requests this Court to enter judgment in Lower48's favor and against Shopify as follows:

- a. finding that Shopify has infringed one or more claims of the '177 patent under 35 U.S.C. §§ 271(a);
- b. finding that Shopify has infringed one or more claims of the '349 patent under 35 U.S.C. §§ 271(a);
- c. finding that Shopify has infringed one or more claims of the '070 patent under 35 U.S.C. §§ 271(a);

- d. finding that Shopify has infringed one or more claims of the '777 patent under 35 U.S.C. §§ 271(a);
- e. awarding Lower48 damages under 35 U.S.C. § 284, or otherwise permitted by law, including supplemental damages for any continued post-verdict infringement;
- f. awarding Lower48 pre-judgment and post-judgment interest on the damages award and costs;
- g. awarding cost of this action (including all disbursements) and attorney fees pursuant to 35 U.S.C. § 285, or as otherwise permitted by the law; and
- h. awarding such other costs and further relief that the Court determines to be just and equitable.

Dated: January 3, 2023

Respectfully submitted,

/s/ Raymond W. Mort, III

Raymond W. Mort, III

Texas State Bar No. 00791308

raymort@austinlaw.com

THE MORT LAW FIRM, PLLC

501 Congress Avenue, Suite 150

Austin, Texas 78701

Tel/Fax: 512-865-7343

Of Counsel:

Ronald M. Daignault (*pro hac vice* to be filed)*

Chandran B. Iyer (*pro hac vice* to be filed)

Oded Burger (*pro hac vice*)*

Tedd W. Van Buskirk (*pro hac vice* to be filed)*

Zachary H. Ellis*

(Texas State Bar No. 24122606)

rdaignault@daignaultiyer.com

cbiyer@daignaultiyer.com

oburger@daignaultiyer.com

tvanbuskirk@daignaultiyer.com

zellis@daignaultiyer.com

DAIGNAULT IYER LLP

8618 Westwood Center Drive - Suite 150

Vienna, VA 22182

Attorneys for Lower48 IP, LLC

*Not admitted to practice in Virginia